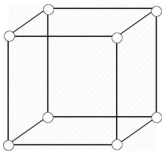
Homework 2

Problem 1: 3-dimensional Boolean functions

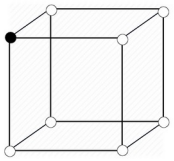
For a function that has k input patterns that map to the target output $t^{(\mu)} = +1$(black ball) and k-8 input patterns that map to $t^{(\mu)} = -1$(white ball):

- k = 0



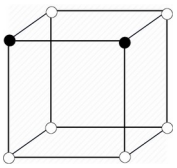
In this case there is only one possible function which is Linearly Separable. In the same way we get one more function for the case k = 8 which is symmetric to this.

- k = 1

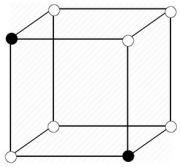


This time we get 8 Linearly Separable functions, one for each different position of the black ball on the corners of the cube. In the same way the symmetric case k = 7 gives us 8 more functions, one for each position of the white ball this time, while the others are black.

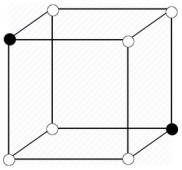
- k = 2



(a)



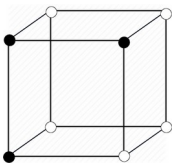
(b)



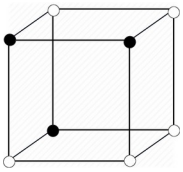
(c)

For k =2, in the case (a), we have 12 different functions which are Linearly Separable, corresponding to the 12 different ways we can put the two black balls as corners of the same edge. The case k = 6 is symmetric to this one so we get another 12 functions from there. For cases (b) and (c) there are no Linearly Separable functions.

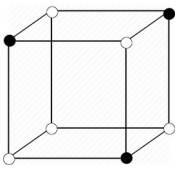
- k = 3



(a)



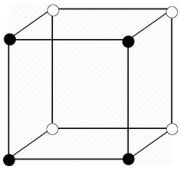
(b)



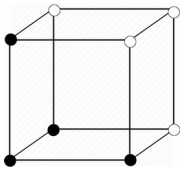
(c)

For the subcase (a) of k = 3 case we find 24 Linearly Separable functions. These functions correspond to 4 different ways we can put the 3 black balls on each side of the cube multiplied by the 6 sides gives us 24 ways to allocate the black balls while the functions are Linearly Separable. For cases (b) and (c) there are no ways to allocate the black balls and result to Linearly Separable functions.

- k = 4

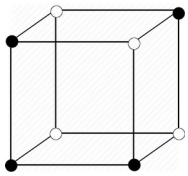


(a)

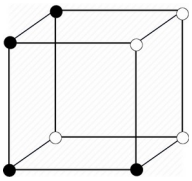


(b)

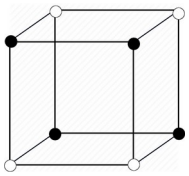
We have 6 Linearly Separable in this case. Each one corresponds to the 6 sides of the cube with black balls on their corners. For case (b) we get 8 more Linearly Separable functions. There are also 4 other subcases for the k = 4 case but none of them, in any way, corresponds to Linearly Separable functions:



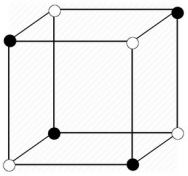
(c)



(d)



(e)



(f)

Thus the total of Linearly Separable functions is 104.

Cases	Number of L.S. Functions
k = 0	1
k = 1	8
k = 2	12
k = 3	24
k = 4	14
k = 5	24
k = 6	12
k = 7	8
k = 8	1

Problem 2: Linear separability of 4-dimensional Boolean functions

For the first problem I wrote the following programm in MATLAB:

```
inputData = csvread('input_data_numeric.csv');
inputData(:,1)=[];
inputDataSize = size(inputData,1);
updates = 10^5;
targets = [[1, 1, 1, -1, 1, 1, 1, 1, -1, -1, -1, -1, 1, 1, -1, -1]
           [1, 1, -1, -1, -1, -1, -1, 1, -1, -1, 1, 1, 1, -1, -1, 1]
           [-1, -1, -1, 1, -1, 1, 1, 1, 1, 1, 1, -1, 1, -1, -1, -1]
           [1, 1, -1, 1, -1, -1, 1, -1, 1, -1, 1, 1, -1, 1, -1, 1]
           [1, -1, 1, 1, 1, 1, -1, -1, 1, -1, 1, 1, 1, 1, -1, 1]
           [-1, 1, -1, -1, -1, -1, -1, 1, -1, -1, -1, -1, -1, 1, -1, -1]
          ];
learningRate = 0.02;
numOfWeights = size(inputData,2);
O = zeros(inputDataSize,1);

for numOfFunc = 1:size(targets,1)

  threshold = 2*rand-1; % [-1,1]
  w = 0.4*rand(numOfWeights, 1)-0.2; % [-0.2, 0.2]

  for j=1:updates
    mu = randi(inputDataSize);
    wSum = 0;
    for k = 1:size(inputData,2)
      wSum = wSum + w(k)*inputData(mu,k);
    end

    b = 1/2*(-threshold+wSum);
    O(mu) = tanh(b);
    if isequal(transpose(sign(O)), targets(numOfFunc,:))
      fprintf('Function %d is linear.\n', numOfFunc);
      break
    end

    wUpdate = learningRate*((targets(numOfFunc,mu)-O(mu))*(1-(tanh(b))^2)*...
      transpose(inputData(mu,:)));
    thresholdUpdate = -learningRate*((targets(numOfFunc, mu)-O(mu))*...
      ((1-tanh(b))^2));
    w = w + wUpdate;
    threshold = threshold + thresholdUpdate;

  end % num of updates

end % num of functions
```

Problem 4: Two-layer perceptron

For the last problem I wrote the following programm in MATLAB:

```
trainingSet = csvread('training_set.csv');
trainingTargets = trainingSet(:, 3);
trainingSet(:, 3) = [];
validationSet = csvread('validation_set.csv');
validationTargets = validationSet(:, 3);
validationSet(:, 3) = [];
trainingSetSize = size(trainingSet,1);
pVal = size(validationSet,1);
learningRate = 0.02;
epochs = 10^2;
iterations = 10^5;
M1 = 8;
M2 = 4;

threshold1 = zeros(M1, 1);
threshold2 = zeros(M2, 1);
threshold3 = 0;

w1 = 0.4*rand(M1, 2)-0.2;
w2 = 0.4*rand(M2, M1)-0.2;
w3 = 0.4*rand(M2, 1)-0.2;

b1 = zeros(M1, 1);
b2 = zeros(M2, 1);
b3 = 0;

V1 = zeros(M1, 1);
V2 = zeros(M2, 1);
O = zeros(trainingSetSize,1);

for ep = 1:epochs

  for i = 1:iterations
    mu = randi(trainingSetSize);
    wSum1 = 0;
    wSum2 = 0;
    wSum3 = 0;

  % Forward Propagation using training set

    for j = 1:M1
      wSum1 = 0;
      for k = 1:2
        wSum1 = wSum1 + w1(j,k)*trainingSet(mu,k);
      end
      b1(j) = wSum1 - threshold1(j);
      V1 = tanh(b1);
```

```
  end

  for l = 1:M2
    wSum2 = 0;
    for k = 1:M1
      wSum2 = wSum2 + w2(l,k)*V1(k,1);
    end
    b2(l) = wSum2 - threshold2(l);
    V2 = tanh(b2);
  end

  for m = 1:M2
    wSum3 = wSum3 + w3(m,1)*V2(m,1);
  end

  b3 = wSum3 - threshold3;
  O(mu) = tanh(b3);

  % Backpropagation

  delta3 = (trainingTargets(mu)-O(mu))*(1-O(mu)^2);
  delta2 = delta3*w3.*(1-V2.^2);
  delta1 = transpose(transpose(delta2)*w2).*(1-V1.^2);

  wUpdate3 = learningRate*delta3*V2;
  thresholdUpdate3 = -learningRate*delta3;
  w3 = w3 + wUpdate3;
  threshold3 = threshold3 + thresholdUpdate3;

  wUpdate2 = learningRate*delta2.*transpose(V1);
  thresholdUpdate2 = -learningRate*delta2;
  w2 = w2 + wUpdate2;
  threshold2 = threshold2 + thresholdUpdate2;

  wUpdate1 = learningRate*delta1.*trainingSet(mu,:);
  thresholdUpdate1 = -learningRate*delta1;
  w1 = w1 + wUpdate1;
  threshold1 = threshold1 + thresholdUpdate1;

end % num of iterations

% Forward propagation using validation set

for mu = 1:pVal
  wSum1 = 0;
  wSum2 = 0;
  wSum3 = 0;

  for j = 1:M1
    wSum1 = 0;
    for k = 1:2
      wSum1 = wSum1 + w1(j,k)*validationSet(mu,k);
```

```
    end
    b1(j) = wSum1 - threshold1(j);
    V1 = tanh(b1);
  end

  for l = 1:M2
    wSum2 = 0;
    for k = 1:M1
      wSum2 = wSum2 + w2(l,k)*V1(k,1);
    end
    b2(l) = wSum2 - threshold2(l);
    V2 = tanh(b2);
  end

  for m = 1:M2
    wSum3 = wSum3 + w3(m,1)*V2(m,1);
  end

  b3 = wSum3 - threshold3;
  O(mu) = tanh(b3);

end

% Error Calculation

sum = 0;
for t = 1:pVal
  sum = sum + abs((sign(O(t))-validationTargets(t)));
end
error = sum/(2*pVal);
fprintf('Validation Error:');
disp(error);

if error<0.12
  csvwrite('w1.csv', w1);
  csvwrite('w2.csv', w2);
  csvwrite('w3.csv', w3);
  csvwrite('t1.csv', threshold1);
  csvwrite('t2.csv', threshold2);
  csvwrite('t3.csv', threshold3);
  break
end

end % num of epochs
```